TRANSFORMATION METHODS FOR OPTIMIZING ELLIPTIC CURVE CRYPTOGRAPHIC COMPUTATIONS

Ins a'>

TECHNICAL FIELD OF INVENTION

The present invention relates to software and hardware implementation of elliptic curve cryptographic systems, in particular, and systems that require computation of calculations involving a finite number of arbitrary field operations within a finite field, in general.

BACKGROUND OF THE INVENTION

In the modern information-based society, the need for global computer and network security is becoming increasingly urgent. Cryptographic systems are fundamental tools used to build systems that ensure privacy, trust, and access control in such diverse areas as electronic commerce, corporate security, digital distribution of intellectual property, and national security, among others.

"Public-key" cryptographic systems, in turn, provide essential capabilities needed in systems requiring secure exchange of information between entities (people or computer systems) that may have never exchanged data with one another before. Most modern information systems, including the Internet, fit this description. As an example, while a consumer may have never had any contacts with a particular on-line vendor, he or she should be able to purchase an item from that vendor in a secure manner. Public-key cryptosystems enable such purchases through providing capabilities such as encryption, decryption, digital signatures, and signature verification. In public-key cryptography, an entity interested in receiving secure messages from others publishes his or her "public key." Others use this public-key to encrypt messages they send to the entity. These messages can be decrypted only through the use of a "private key" which is known only to the entity. The entity can also use this private key to digitally "sign" a piece of data.

Others, in turn, can use the public key to verify the signature and ascertain that the data was indeed signed by the signing entity.

The security of a public-key cryptosystem depends on how difficult it is to derive a private key from its associated, known public key. The more complex it is to mathematically derive the private key, the more time it takes a computer to "break" a public key by "guessing" its corresponding private key. Today, the most commonly used public-key cryptography system is the RSA public-key cryptosystem. The relationship between RSA's public and private keys is governed by the mathematics of factorization of large composite integers. RSA public and private keys are large integers represented as a binary bit pattern. The longer a key, the harder it is and the longer it takes a computer to break it by deriving its private key. For example, modern advances in factorization algorithms and distributed computing have made breaking 400-bit RSA keys possible. Breaking an RSA key of length 1024 or 2048 bits, however, is thought to be virtually impossible given the computing resources available today. To retain an acceptable level of security, modern systems have been using longer RSA keys. Since performing public-key cryptography using longer keys requires more computing resources, it is economically ideal to use an alternative public-key cryptosystem that provides the same level of security with shorter keys.

Over the last decade, Elliptic Curve Cryptography ("ECC") has emerged as one possible alternative for an effective cryptosystem. ECC offers the same level of security as RSA with keys that are one-sixth the length of RSA keys. Until now, however, existing software implementations of ECC have been too inefficient to be commercially viable. In order to be commercially viable, ECC needs to allow the same functionality as RSA at comparable speeds, as well as lower costs of implementation in hardware and software. Efficient ECC will enable implementation of many envisioned modern systems that would otherwise be economically infeasible. As such, much research has been focused on achieving efficient ECC in the academia and industry. The most common approach to achieving efficient ECC is briefly described below.

To perform public-key cryptography, ECC methods take advantage of specific features of mathematical "groups" called "elliptic curves." An elliptic curve is related to and "constructed over" a mathematical "field." Any finite field can be chosen to construct an elliptic curve, but the exact choice of the field significantly affects the properties of the elliptic curve and the efficiency of computer implementations that represent the "operations" defined within that elliptic curve. One of the most computationally intense operations used in all ECC implementations is known as "elliptic curve point multiplication." Point multiplication requires the computation of eP, where P is a "point" in the elliptic curve and e is a positive integer. This operation is central to many elliptic curve cryptography functions, including encryption, decryption, random number generation, key-exchange, digital signing, and signature verification.

Over the last decade, a debate has been carried on in the cryptography community over which categories of fields provide the best choices for use with ECC. Two broad categories of fields, called $GF(p)$ and $GF(2^k)$ have been chosen by the Institute of Electrical and Electronics Engineers (IEEE) as international standards for Elliptic Curve Cryptography. While most academic and commercial research today is concentrated on implementing ECC over either $GF(p)$ or $GF(2^k)$, the exact advantages or disadvantages of each choice with respect to cryptography is not clearly understood at this point. Furthermore, both $GF(p)$ and $GF(2^k)$ encompass countless particular individual member fields within them. Each individual member field has its own properties that affect the computational characteristics of an ECC implementation. Furthermore, given a particular individual member field within $GF(p)$ or $GF(2^k)$, numerous elliptic "curves" can be constructed over such field. The choice of the curve, too, affects the computational characteristics of the resulting ECC implementation.

Prior attempts for creating efficient ECC implementations have usually been based on finding either specific individual member fields in $GF(2^k)$ or $GF(p)$ or specific curves defined over such individual member fields which possess "special" mathematical or computational properties.

These special properties would then be exploited to optimize ECC computations. This approach does not attempt to achieve efficient ECC across all mathematical fields. Rather, it concentrates on carefully choosing a particular field so that a specific mathematical or computational technique can be deployed to achieve efficient ECC computations.

An example of this is the proposal by Agnew, et al to utilize a method called "Normal Basis" to achieve fast ECC in particular fields in $GF(2^k)$. While, most academic and industry research has focused on using an alternate method known as "Polynomial Basis" for fields in $GF(2^k)$, the use of Normal Basis in six particular fields within $GF(2^k)$ has allowed commercialization of a particular implementation of ECC. One disadvantage of this approach is that key lengths are limited to the six values allowed by those particular fields.

MATHEMATICAL BACKGROUND OF THE INVENTION

This Section presents a list of some of the mathematical terms that are used in this document. Some key concepts that can be useful in following the methodology of the invention are also briefly described in this Section. The descriptions in this Section are not meant to be mathematically precise or rigorous.

Sets

A "set" is any collection of objects, including mathematical and physical objects. Often, a set is represented in print by enclosing a comma-separated list of the objects that make up the set within the curly brackets, "{" and "}". For example, let F represent the set of all non-negative integers smaller than 7. That set can be written as $F = \{0, 1, 2, 3, 4, 5, 6\}$. An object that belongs to a set is a "member" or "element" of the set. In another example, F denotes the set of all polynomials of order 4, and $p(x)$ represents the specific polynomial $x^4 + x^2 - x + 1$. Since $p(x)$ is a polynomial of order 4, then $p(x)$ is an element of F. In mathematical shorthand, $p(x) \in F$, where

the symbol "$\in$" is commonly read as "belongs to" or "is a member of." A particular set S is a "subset" of another set F if every element of the set S is also an element of the set F. This is denoted by the shorthand notation $S \subset F$. For instance $\{5, 1, 3\} \subset \{0, 1, 2, 3, 4, 5, 6\}$. The "$\subset$" symbol is read as "is a subset of." Every set is a subset of itself: Given any set S, then $S \subset S$.

5

Mappings

A "mapping" is a relationship that associates each member of a set with a particular member of another set. For instance, T can be defined as the relationship that "maps" each member of the set of all human beings to the integer that represents that person's age. If Tom is 32 years old, then $T(\textit{Tom}) = 32$ is written to denote the relationship that T establishes between the integer 32 and the human being Tom. 32 is said to be the "image" of Tom "under" the mapping T.

In another example, let p be a prime number, n denote any non-negative integer, and r denote the integer remainder which results when n is divided by p. A mathematical shorthand for this is $r = n \bmod p$. For instance, if $p = 7$ and $n = 15$, then $r = 15 \bmod 7 = 1$, which is the remainder of 15 divided by 7, since $15 = 2 \cdot 7 + 1$. A mapping T may be constructed between the set of all non-negative integers $N = \{0, 1, 2, 3, \ldots\}$ and the set $R = \{0, 1, 2, 3, 4, 5, 6, 7, 8\}$ in the following manner: given that $p = 7$ and given any non-negative integer $n \in N$, let $T(n) = r = n \bmod p$. Thus, $T(37) = r = 37 \bmod 7 = 2$. Note that regardless of what value n takes, $T(n)$ is an integer less than 7. In other words, given any $n \in N$, then $T(n) \in R$. By convention, T is said to map the set N "into" the set R. This is denoted in shorthand as $T: N \rightarrow R$, which is read, "T is a mapping from the set N into the set R." N is referred to as the "domain" of the mapping T, while R is said to be the "range" of the mapping T.

The "image" of the set N under the mapping T is the unique subset of R where every

element is an image of at least one element of N. In other words, if F denotes the image of N under T, then, given any element $y \in F$, there exists at least one element $x \in N$, such that $T(x) = y$. Since the remainder of the division of any positive integer by 7 is one of the numbers from 0 to 6, the image of N under T in the above example is the set $F = \{0, 1, 2, 3, 4, 5, 6\}$. Since $F \subset R$ (recall that $R = \{0, 1, 2, 3, 4, 5, 6, 7, 8\}$), and no element of N is mapped to any element of R outside of F, then T is also a mapping from N into F. In other words, $T: N \rightarrow F$. Since every member of F is an image of some element of N under T, then T is said to map N "onto" F. Sometimes, the word "transformation" is used to refer to a mapping.

Set Operations

An "ordered pair" is a mathematical notion that references pairs of objects under circumstances where one needs to keep track of which object is the "first" element of the pair and which object is the "second" element of the pair. For instance, the set of all pairs of husbands and wives is a set of ordered pairs, whose members can be represented by the notation (x, y), where x is an element of the set of all husbands, and y is an element of the set of all wives. Let X and Y be any arbitrary sets. The "cross product" of X and Y is the set of all ordered pairs whose first elements come from X and whose second elements come from Y. In mathematical parlance, the cross product of X and Y is written as $X \times Y$ and is defined by the set of all ordered pairs (x, y), where $x \in X$, and $y \in Y$. As an example, let $X = \{0,1\}$ and $Y = \{0,1,2\}$, then $X \times Y = \{(0, 0), (0, 1), (0, 2), (1, 0), (1, 1), (1, 2)\}$

Given any set S, then a mapping from $S \times S$ into S can be referred to as a "binary set operation" defined within S (the word *binary* underscores the fact that each element of the domain of the mapping is an ordered *pair*.) For instance, let $F = \{0, 1, 2, 3, 4, 5, 6\}$. Next, construct a mapping $T: F \times F \rightarrow F$ as follows: given any ordered pair $(x, y) \in F \times F$, where $x \in F$

and $y \in F$, let the image of (x, y) under T be the integer that is the result of calculating the expression $(x + y) \bmod 7$. In other words, let $T((x, y)) = (x + y) \bmod 7$. It is relatively straightforward to verify that the range of T is in fact the set F. For example, $T((4, 6)) = (4 + 6) \bmod 7 = 10 \bmod 7 = 3$. Regardless of the value of $x + y$, the integer which is the result of the expression $(x + y) \bmod 7$ is the remainder of a division by 7 and therefore an integer between 0 and 6, and a member of F. Hence, T is a binary set operation defined within F.

For the sake of convenience, when working with a given binary operation T, the construct $T((x, y))$ is often written as $x \cdot y$. The symbol "$\cdot$" is called the "binary operator" and is used to represent the binary operation T. For instance, given the above definition for T, instead of writing $T((4, 6)) = 3$, one writes $4 \cdot 6 = (4 + 6) \bmod 7 = 3$. Other symbols may also be used as binary operator symbols. Two other commonly used such symbols are + and $\oplus$. The two members of the set S that make up the ordered pair that a binary operation maps into another member of the set S are the "operands" in the operation, which, in turn, is said to "operate on" the operands. For example, in the equation, $4 \cdot 6 = 3$, the operands are 4 and 6.

Groups

A "group" is a set G together with a binary operation "$\cdot$" defined within the set G such that the following three conditions are satisfied:

(i) Given $x, y, z \in G$, then $x \cdot (y \cdot z) = (x \cdot y) \cdot z$. This is known as the associative property of the group.

(ii) There exists a unique element $I \in G$, such that $x \cdot I = I \cdot x = x$, for all $x \in G$. The element I is referred to as the "identity" element in G.

(iii) Given any $x \in G$, there exists an element $x^{-1} \in G$, such that $x \cdot x^{-1} = I$. The element x^{-1} is referred to as the "inverse" of x under the $\cdot$ operation.

The · operation is referred to as the "group operation." It is the existence of the group operation · defined within the set G that allows G to be a group. In fact, G is said to be group under the · operation. An "Abelian" group is a group G such that given any $x, y \in G$, then $x \cdot y = y \cdot x$.

As an example of an Abelian group, consider the set $F = \{1, 2, 3, 4, 5, 6\}$ together with the operation · given by or defined as $x \cdot y = (x \cdot y) \bmod 7$ for all elements $x, y \in F$ (the second "·" represents the common operation of integer multiplication.) It is known that F is a group under this "multiplication operation." To demonstrate this, Table 1, which contains the values of $x \cdot y = (x \cdot y) \bmod 7$ for all possible combinations of elements $x, y \in F$, is constructed below. To look up the value of $x \cdot y$ using the table, locate the cell that is at the intersection of the row whose label is the value of x with the column whose label is the value of y.

Table 2. The Multiplication Operation in $F = \{1, 2, 3, 4, 5, 6\}$

Given by $x \cdot y = (x \cdot y) \bmod 7$

	1	*2*	*3*	*4*	*5*	*6*
1	1	2	3	4	5	6
2	2	4	6	1	3	5
3	3	6	2	5	1	4
4	4	1	5	2	6	3
5	5	3	1	6	4	2
6	6	5	4	3	2	1

As an example, note that if $x = 5$ and $y = 6$, the table gives $x \cdot y = 2$. To verify the accuracy of this, note that $x \cdot y = 5 \cdot 6 = (5 \cdot 6) \bmod 7 = 30 \bmod 7 = 2$.

Given the above definitions for · and F, the task of verifying that F is a group is tantamount to verifying that conditions (i), (ii), and (iii) above are satisfied:

(i) Condition (i) provides that $x \cdot (y \cdot z) = (x \cdot y) \cdot z$. Using x=3, y=5 and z=2, consider that $3 \cdot (5 \cdot 2) = 3 \cdot ((5 \cdot 2) \bmod 7) = 3 \cdot (10 \bmod 7) = 3 \cdot 3 = (3 \cdot 3) \bmod 7 = 9 \bmod 7 = 2$; and that $(3 \cdot 5) \cdot 2 = ((3 \cdot 5) \bmod 7) \cdot 2 = (15 \bmod 7) \cdot 2$

$1 \cdot 2 = (1 \cdot 2) \bmod 7 = 2 \bmod 7 = 2$. Therefore, $(3 \cdot 5) \cdot 2 = 3 \cdot (5 \cdot 2)$.

(ii) Table 1 demonstrates that the identity element of F under the group operation $\cdot$ is 1. For, as the table shows, $x \cdot 1 = 1 \cdot x = 1$ for all $x \in F$.

(iii) Using Table 1, the following values for the inverse, x^{-1}, of each element x of F (where $x \cdot x^{-1} = 1$) can be derived: $1^{-1} = 1$, $2^{-1} = 4$, $3^{-1} = 5$, $4^{-1} = 3$, $5^{-1} = 3$, and $6^{-1} = 1$.

(iv) Examining Table 1 also establishes that for all $x, y \in F$, $x \cdot y = y \cdot x$. Hence, F is an Abelian group under the multiplication operation.

Sometimes, the symbol + is used to denote the group operation in F. In such cases, the inverse of any element $x \in F$ under + is denoted by $-x$ rather than x^{-1}. As another example of an Abelian group, consider the set $F = \{0, 1, 2, 3, 4, 5, 6\}$ together with the operation + given by or defined as $x + y = (x + y) \bmod 7$ for all elements $x, y \in F$ (the second "+" represents the common operation of integer addition.) It is known that F is a group under this "addition operation." To demonstrate this, Table 2, which contains the values of $x + y = (x + y) \bmod 7$ for all possible combinations of elements $x, y \in F$, is constructed below.

Table 2. The Addition Operation in $F = \{0, 1, 2, 3, 4, 5, 6\}$

Given by $x + y = (x + y) \bmod 7$

	0	***1***	***2***	***3***	***4***	***5***	***6***
0	0	1	2	3	4	5	6
1	1	2	3	4	5	6	0
2	2	3	4	5	6	0	1
3	3	4	5	6	0	1	2
4	4	5	6	0	1	2	3
5	5	6	0	1	2	3	4
6	6	0	1	2	3	4	5

As an example, note that if $x = 5$ and $y = 6$, the table gives $x + y = 4$. To verify the accuracy of this, note that $x \cdot y = 5 \cdot 6 = (5 \cdot 6) \bmod 7 = 11 \bmod 7 = 4$.

Given the above definitions for + and F, the task of verifying that F is a group is tantamount to verifying that conditions (i), (ii), and (iii) above are satisfied.

(i) As an example of condition (i) holding, consider that $(3 + 5) + 2 = ((3 + 5) \bmod 7) + 2 = (8 \bmod 7) + 2 = 1 + 2 = (1 + 2) \bmod 7 = 3 \bmod 7 = 3$; and that $3 + (5 + 2) = 3 + ((5 + 2) \bmod 7) = 3 + (7 \bmod 7) = 3 + 0 = (3 + 0) \bmod 7 = 3 \bmod 7 = 3$. Therefore, $(3 + 5) + 2 = 3 + (5 + 2)$.

(ii) Table 2 demonstrates that the identity element of F under the group operation + is 0. For, as the table shows, $x + 0 = 0 + x = 0$ for all $x \in F$.

(iii) Using Table 2, the following values for the inverse, $-x$, of each element x of F (where $x + (-x) = 0$) can be derived: $-0 = 0$, $-1 = 6$, $-2 = 5$, $-3 = 4$, $-4 = 3$, $-5 = 2$, and $-6 = 1$.

(iv) Examining Table 2 also establishes that for all $x, y \in F$, $x + y = y + x$. Hence, F is an Abelian group under the addition operation.

Fields

A "field" is a set F together with two binary set operations + and · defined within F such that the following conditions are met:

(i) F is an Abelian group under the + operation. The + operation is referred to as the "addition operation" of the field. The identity element of the field under the addition operation is denoted as 0. Given any element $x \in F$, the inverse of x under the addition operation of the field is denoted by $-x$, which is referred to as the "additive inverse" of x.

(ii) If 0 were to be removed from the set F, the resulting set would be an Abelian group under the · operation. The · operation is referred to as the "multiplication

operation" of the field. The identity element of the field under the multiplication operation is denoted as 1, which is an element of F distinct from 0. Given any element $x \in F$, the inverse of x under the multiplication operation of the field is denoted by x^{-1}, which is referred to as the "multiplicative inverse" of x.

(iii) Given any $x \in F$, then $x \cdot 0 = 0 \cdot x = 0$.

(iv) Given any $x, y, z \in F$, then $x \cdot (y + z) = (x \cdot y) + (x \cdot z)$. This is known as the "distributive property" of the field.

An example of a field is the set $F = \{0, 1, 2, 3, 4, 5, 6\}$ together with the + operation defined as $x + y = (x + y) \bmod 7$ and the $\cdot$ operation defined as $x \cdot y = (x \cdot y) \bmod 7$ for all $x, y \in F$. It is known that F forms a field under these addition and multiplication operations. To demonstrate this fact, it is necessary to show that the four conditions described above are satisfied. Conditions (i) and (ii) where shown to be satisfied in the previous Section. Condition (iii) is evident from the Table 3, below.

Table 3. The Multiplication Operation in $F = \{0, 1, 2, 3, 4, 5, 6\}$

	0	*1*	*2*	*3*	*4*	*5*	*6*
0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6
2	0	2	4	6	1	3	5
3	0	3	6	2	5	1	4
4	0	4	1	5	2	6	3
5	0	5	3	1	6	4	2
6	0	6	5	4	3	2	1

As an example of condition (iv) holding, we will show that $3 \cdot (5 + 6) = (3 \cdot 5) + (3 \cdot 6)$. Indeed, $3 \cdot (5 + 6) = (3 \cdot ((5 - 6) \bmod 7)) \bmod 7 = (3 \cdot (11 \bmod 7)) \bmod 7 = (3 \cdot 4) \bmod 7 = 12 \bmod 7 = 5$, while, $(3 \cdot 5) + (3 \cdot 6) = (((3 \cdot 5) \bmod 7) - ((3 \cdot 6) \bmod 7)) \bmod 7 = ((15 \bmod 7) - (18 \bmod 7)) \bmod 7 = (1 + 4) \bmod 7 = 5 \bmod 7 = 5$, too.

A field F is a "finite field" if it has a finite number of elements. The field F above is a

specific example of a family of finite fields known as *GF(p)*, where *p* is any prime number. Given a particular prime number *p*, *GF(p)* is defined as the set *{0, 1, ..., p–1}* of non-negative integers less than *p*, together with the addition operation + given by integer addition *mod p*, and the multiplication operation · given by integer multiplication *mod p*. The field *F* used in the above example is the field *GF(7)*.

Field Arithmetic

The mathematical concept of fields is an abstraction of the familiar "rational" number system. The rationals are the set of all integers together with all numbers that can be represented as a fraction whose nominator and denominator are both non-zero integers. The set of all rational numbers can in fact be shown to be a field under the common operations of addition and multiplication of fractions. As such, common mathematical techniques of arithmetic have also been carried over to the more abstract domain of fields. Given a particular field *F*, mathematical field theory allows the writing and evaluation of "valid" arithmetic expressions and equations whose constants and variables "come from the field, " *i.e.* are members of *F*.

For instance, consider the equation $y = (2 \cdot x) + 1$. The set of all ordered pairs *(x, y)* of *rational* numbers that satisfy this equation includes such elements as *(0, 1)*, *(2, 5)*, *(4, 9)*, and *(3.45, 7.9)*. However, since the constants in this equation, *2* and *1*, are also members of the field *GF(7)*, the "expression" $(2 \cdot x) + 1$ is a valid expression in GF(7), meaning that as long as the value that is substituted for *x* in the expression is a member of *GF(7)*, it is guaranteed that the expression will "evaluate" to a valid member of *GF(7)*. As such, the equation itself is a valid equation in *GF(7)*, too. In fact, the set of all "solutions" to this equation, i.e. the set of all ordered pairs that satisfy the equation in *GF(7)*, is equal to *{(0, 1), (1, 3), (2, 5), (3, 0), (4, 2), (5, 4), (6, 6)}*.

To facilitate working with more complicated expressions, a few mathematical shorthands are utilized in field arithmetic. Given any field F, any element $x \in F$, and any positive integer k, the expression x^k represents that unique element of F which results when x is multiplied by itself k times using the multiplication operation in F. In other words $x^k = x \cdot x \cdot \ldots \cdot x$, where there are $k-1$ many $\cdot$ operators in the expression. By convention, x^0 is defined to be equal to 1.

Given any integer k and any element x in F, the expression kx represents that unique element of F which results when x is added to itself k many times using the addition operation in F. In other words, $kx = x + x + \ldots + x$, where there are $k-1$ many + operators in the expression.

Given c a constant in F and x a variable defined over F, the expression $c \cdot x$ is commonly written as cx. Furthermore, given x and y, any two elements of F, the expression $x + (-y)$ is commonly written as $x - y$.

Given a field F and any element x in F, the task of computing x's additive inverse, $-x$, or x's multiplicative inverse, x^{-1}, may be computationally intense. It is possible to view the task of computing the inverse of x as a set operation. A "unary set operation" T defined within a set S is a mapping from S onto S. The word *unary* underscores the fact that unlike binary set operations, the domain of T is made up of single, individual members of S. Given any element x of S, let T map x to x^{-1}. In other words, let $T(x) = x^{-1}$. Then T is a unary set operation.

Given a particular field F and an integer k, a polynomial $p(x)$ of order k defined over F is an expression of the form $p(x) = a_k x^k + a_{k-1} x^{k-1} + \ldots + a_1 x + a_0$. In this definition, x is a variable in F, meaning that before the expression is evaluated, some particular element of F must be substituted for x in the expression. The particular member of F which is substituted for x is the value that x is "bound" to. The a_i's ($0 \leq i \leq k$) are known as the "polynomial coefficients" of $p(x)$ and are constants in F, meaning that they are chosen before a value for x is selected, and that regardless of the value that x takes on, the values of the polynomial coefficients remain the same.

When $F = GF(2)$, the set of all polynomials of degree k defined over $GF(2)$ is referred to as $GF(2^k)$. It is known that given any k greater than 1, specific addition and multiplication operations can be defined within $GF(2^k)$ in such a way so that $GF(2^k)$ forms a field under such operations. The set $GF(2^k)$ is the set of all polynomials of order k whose polynomial coefficients are either 0 or 1. For instance, $p(x) = x^5 + x^2 + 1$ is a member of $GF(2^5)$ whose polynomial coefficients are given by $a_5 = 1$, $a_4 = 0$, $a_3 = 0$, $a_2 = 1$, $a_1 = 0$, and $a_0 = 1$.

Optimizing Field Arithmetic Calculations

In addition to the fields of information security and cryptography, there are numerous other problems in business and science, which are either based on or utilize the mathematics of finite fields. Computer applications dealing with such problems often need to carry out calculations involving finite field arithmetic. This often takes the form of evaluating a mathematical expression f involving a finite number of constants, variables, coefficients and operations defined within a finite field F. Note that variables and constants must be members of the field F, but coefficients may take any integer value. Coefficients, such as the 5 in the expression $x - 5y$, are not elements of the field F, and merely represent a shorthand notation for repeated addition, in this case $5y = 2y + 2y + y$, where $2y = y + y$. Since computational efficiency is of concern, we will assume that if the same quantity occurs in more than one part of an expression, such as $(x_2 - x_1)^{-1}$ does above, each such quantity is only computed once. Also, without loss of generality, the expression f can and will be assumed to be in fully reduced form, in which all calculations in the expression that involve only constants have already been performed, and the resulting constants substituted into the expression.

As an example of such an expression, let $F \in GF(p)$, let x_1, x_2, y_1, y_2 be variables defined in F, let a be a constant that is some element from F, and define,

$$f = ((y_2 - y_1) \cdot (x_2 - x_1)^{-1}) \cdot ((y_2 - a) \cdot (x_2 - x_1)^{-1}) - x_1 - 5\,x_2$$

Since the only variables and constants in the expression f are x_1, x_2, y_1, y_2 and a, the expression f involves a finite number of field elements. Furthermore, the expression f involves four unary additive inversion operations to calculate $-x_1$, $-x_2$, $-y_1$, and $-a$, a single unary multiplicative inversion operation to calculate $(x_2 - x_1)^{-1}$, three binary multiplication operations, three binary addition operations to calculate $5\,x_2$, four binary addition operations to calculate the expressions in parentheses, and two binary addition operations to calculate $(\ldots) - x_1 - 5\,x_2$. Hence, the expression f involves a finite number of field operations, too. Consequently, when the expression f is evaluated, that is, when the calculations specified in the expression f are carried out, the result of the calculation is going to be a single element in the field F.

Any given expression f defined within a finite field F is usually composed of "subexpressions." Any part of the expression f which by itself is a valid expression in F is a subexpression of f. For instance, the expression $s = (x_2 - x_1)^{-1}$ is a valid expression in F, if both x_2 and x_1 are members of F. Therefore $s = (x_2 - x_1)^{-1}$ is a subexpression of the expression $f = ((y_2 - y_1) \cdot (x_2 - x_1)^{-1}) \cdot ((y_2 - y_1) \cdot (x_2 - x_1)^{-1}) - x_1 - x_2$. Some of the other subexpressions of f are $s = x_1$, $s = -y_1$, and $s = (y_2 - y_1) \cdot (x_2 - x_1)^{-1}$. Note, however, that and $s = x_2 - x_1)^{-1}) - x_1$ is not a subexpression of f, because s is not a valid expression in F. Every expression f is a subexpression of itself.

Given an expression f defined within a finite field F, the task of evaluating the expression f can be computationally intense. Techniques that allow efficient calculation of such expressions in computer software and/or hardware may have significant business and scientific value. Given the exact nature of the applications such calculations occur in, different criteria may be used to determine what exactly constitutes an "efficient" calculation. In certain applications, it may be desirable to optimize calculations so that higher computation speeds are achieved. In other

applications, it may be important to optimize for minimal use of silicon area in hardware implementations. Still other applications may benefit from optimization that allow parallel computation of the calculations. In particular, the inversion operation, which in the fields $GF(p)$ and $GF(2^k)$ uses the Fermat method, is computationally very intensive. To avoid this, methods of formulating problems in "projective coordinates" have been developed, by Menezes and others, which allow calculations to be reformulated in a manner that removes the need to perform any inversion operations, usually at the expense of increasing the number of other operations.

The Montgomery Algorithm

In 1985, P. L. Montgomery published an algorithm which can be used to optimize the task of computing an expression of the form $f = a \cdot b \cdot r^{-1}$, where a, b, and r are elements of a field $F \in GF(p)$ and $\cdot$ is the multiplication operation in F. Since 1985, some work in industry and academia has been focused on extending the use of the Montgomery algorithm to expressions of a more general form than $f = a \cdot b \cdot r^{-1}$. To facilitate discussion of such efforts, this patent defines the term "Montgomery Canonical Form." Given a particular element r of a field F, an expression f in F is recursively defined to be in the Montgomery Canonical Form with respect to r as such,

(i) An expression f is in the Montgomery Canonical Form with respect to r, if it does not contain any field multiplication operations and also does not contain r. That is, if no subexpression s exists of the form $s = s_1 \cdot s_2$, where s_1 and s_2 are subexpressions of f. For example, the expression $f = (x_1 - x_1 - a)$ and the expression $f = x_1$ are both in the Montgomery Canonical Form with respect to r.

(ii) An expression f is in the Montgomery Canonical Form with respect to r, if it can be written in the form of $f = f_1 \cdot f_2 \cdot r^{-1}$, where f_1 and f_2 are both subexpressions of f which are themselves in the Montgomery Canonical Form with respect to r. Note that to

determine whether or not an expression is in Montgomery Canonical Form with respect to r, the expression is to be considered in isolation. For example, the expression $f = (x_1 + x_1 + x_1) \cdot x_1 \cdot r^{-1}$ is in the Montgomery Canonical Form with respect to r. However, $f = x_1 \cdot (x_2 \cdot x_2) \cdot r^{-1}$ is not in the Montgomery Canonical Form with respect to r, since although $(x_2 \cdot x_2) \cdot r^{-1}$ is in the Montgomery Canonical Form with respect to r, the single factor of r^{-1} cannot simultaneously be considered to be part of the subexpression $(x_2 \cdot x_2) \cdot r^{-1}$ and also of the whole expression, so $(x_2 \cdot x_2)$ is not in the Montgomery Canonical Form with respect to r.

(iii) An expression f is in the Montgomery Canonical Form with respect to r, if it can be written in the form of $f = (s)^{-1} \cdot r^2$ where s is a subexpression of f which is itself in the Montgomery Canonical Form with respect to r. For example, the expression $f = (x_2 - x_1)^{-1} \cdot r^2$ is in the Montgomery Canonical Form with respect to r. And finally,

(iv) The expression f is in the Montgomery Canonical Form with respect to r, if whenever there exists a subexpression s of f which can be written as $s = s_1 \cdot s_2$, where s_1 and s_2 are subexpressions of f which are both in the Montgomery Canonical Form with respect to r, then there exists a unique subexpression s_3 of f such that $s_3 = s_1 \cdot s_2 \cdot r^{-1}$. For example, the expression $f =$

$$(((x_1 - x_1 + x_1) \cdot x_1 \cdot r^{-1} + a) \cdot z \cdot r^{-1}) \cdot (((x_1 + x_1 + x_1) \cdot x_1 \cdot r^{-1} + a) \cdot z \cdot r^{-1}) \cdot r^{-1} - x_1 - x_1$$

is in the Montgomery Canonical Form with respect to r.

Given a field $F \in GF(p)$ and an element $r \in F$, the Montgomery algorithm can be applied effectively to optimize computation of any expression f in F which is in the Montgomery Canonical Form with respect to r. Given an arbitrary expression f, then, there may be efficiencies gained by "transforming" the expression f into some other expression f' (read as "f prime") which is in Montgomery Canonical Form. During the past decade and a half, an innumerable number of

expressions involving a finite number of operations within finite fields in *GF(p)* have been encountered within the confines of specific applications in business and academia. In some instances, researchers and engineers have transformed certain such expressions into other expressions which are in the Montgomery Canonical Form, and which are therefore faster to compute. Until the present invention, however, no general method for transforming *any* arbitrary expression involving a finite number of field operations in *GF(p)* into an expression that is in Montgomery Canonical Form with respect to some r in *GF(p)* has been known.

A particular expression that is commonly encountered in business and academic applications involving fields in *GF(p)* is one of the form $f = x^k$, where k is some positive integer and x is an element in $F \in GF(p)$. Calculating $f = x^k$ is known as the exponentiation of x. It is well known that a particular "substitution technique" described in the next section in this document can be applied to the expression $f = x^k$ to transform it into another expression, f', which is in Montgomery Canonical Form with respect to some particular element r in F. The Montgomery algorithm is commonly applied to the resulting expression f' to provide an efficient method for exponentiation of x.

Although the Montgomery algorithm has been used for over a decade for fast exponentiation in *GF(p)*, no method for extending its speed improvements to general finite field calculations has been available until the present invention.

In 1998, C. K. Koc and T. Acar published an algorithm which can be used to optimize the task of computing an expression of the form $f = a \cdot b \cdot r^{-1}$, where a, b, and r are elements of a field $F \in GF(2^k)$ and $\cdot$ is the multiplication operation in F. This algorithm is referred to as the Montgomery Algorithm in $GF(2^k)$. The Montgomery Algorithm in $GF(2^k)$ has been applied in the past to speed up exponentiation in $GF(2^k)$ in a manner analogous to the method used for speeding up exponentiation in *GF(p)*. Until the present invention, however, no method for extending the

speed improvements of this algorithm to general finite field calculations has been available.

Substitution Technique

This section describes a particular substitution technique that is often used to manipulate expressions involving elements and operations defined within a field F. The technique involves replacing all instances of a specific pattern of operations and/or operands in f with another specific pattern. As an example, let x and y represent any member of F and let a, b, c, and r be specific elements in F. Then, if all occurrences of the pattern $x \cdot y$ in $f = (a \cdot b) \div (c \cdot b)$ are replaced by the pattern $x \cdot y \cdot r$, the resulting expression f' is given by $f' = (a \cdot b \cdot r) \div (c \cdot b \cdot r)$. To facilitate discussion of the technique, let s be an expression that represents the pattern that is to be replaced. The expression s is called the "source" expression. Let t be an expression that represents the pattern that s is replaced with. The expression t is called the "target" expression. The rest of this section describes how the current substitution technique is applied to three simple types of source expressions.

Case 1. The source expression s involves no operators

In this case, the source expression s is given by $s = x$, where x stands for any single variable within the expression f. The substitution technique simply replaces all occurrences of the variable represented by the source expression s by the pattern given by the target expression t.

Case 2. The source expression involves a single unary operator

In this case, the source expression is given either by $s = -x$ or $s = x^{-1}$, where x stands for any subexpression of the expression f. Here, the substitution technique calls for constructing the set S of all subexpressions of f that "match" the source expression s. In other words, the set S is

given by the set of all subexpressions s of the expression f which are of the form $s = \cdot x$ or $s = x^{-1}$, where x is itself a subexpression of f. Note that given any two subexpressions of f in the set S, one may be a subexpression of the other. The substitution technique works by replacing each member of the set S by the corresponding pattern given by the target expression t, except that before the substitution technique is applied to any member s of the set S, it is first applied to any other members of S that s is a subexpression of.

Case 3. The source expression involves a single binary operator

In this case, the source expression is given either by $s = x + y$ or $s = x \cdot y$, where x and y stand for any subexpressions of the expression f. Here, the substitution technique calls for constructing the set S of all subexpressions of f that "match" the source expression s. In other words, the set S is given by the set of all subexpressions s of the expression f which are of the form $s = x + y$ or $s = x \cdot y$, where x and y are themselves subexpressions of f. Note that given any two subexpressions of f in the set S, one may be a subexpression of the other. The substitution technique works by replacing each member of the set S by the corresponding pattern given by the target expression t, except that before the substitution technique is applied to any member s of the set S, it is first applied to any other members of S that s is a subexpression of.

A substitution technique similar to this has been used in business and industry in the past in a two-step process to transform instances of simple expressions, of the form $f = x^k$, into another expression f' which is in Montgomery Canonical Form. To illustrate an example, this will be demonstrated for the case when $k = 4$, which means that the expression f is given by $f = x^4 = x \cdot x \cdot x \cdot x = ((x \cdot x) \cdot x) \cdot x$.

(i) Let the source expression s be given by $s \quad x \cdot y$. Let the target expression t be given by t

$= x \cdot y \cdot r^{-1}$, where r is a constant in the field F. Applying the substitution technique to the expression $f = ((x \cdot x) \cdot x) \cdot x$ yields the expression $f' = ((x \cdot x \cdot r^{-1}) \cdot x \cdot r^{-1}) \cdot x \cdot r^{-1}$. Note that the expression f' is in the Montgomery Canonical Form, since every subexpression of f' enclosed in parenthesis is in the Montgomery Canonical Form with respect to r. Because of this, the following step can allow the Montgomery algorithm to be applied to efficiently calculate f'.

(ii) Let the source expression s be given by $s = x$, where x stands for a variable or constant in f. Let the target expression t be given by $t = x \cdot r$. Applying the substitution technique to replace every occurrence of x with $x \cdot r$ in the expression $f_1 = ((x \cdot x \cdot r^{-1}) \cdot x \cdot r^{-1}) \cdot x \cdot r^{-1}$ yields the expression $f' = (((x \cdot r) \cdot (x \cdot r) \cdot r^{-1}) \cdot (x \cdot r) \cdot r^{-1}) \cdot (x \cdot r) \cdot r^{-1}$..

This substitution technique allows efficient computation of $f = x^k$, because it can be shown that the expression f is equivalent to $f = f' \cdot r^{-1}$. To see this in the case $k = 4$, note that

$$f' = (((x \cdot r) \cdot (x \cdot r) \cdot r^{-1}) \cdot (x \cdot r) \cdot r^{-1}) \cdot (x \cdot r) \cdot r^{-1}$$

$$= ((x \cdot r \cdot x \cdot r \cdot r^{-1}) \cdot x \cdot r \cdot r^{-1}) \cdot x \cdot r \cdot r^{-1}$$

$$= ((r \cdot x \cdot x \cdot r \cdot r^{-1}) \cdot x \cdot r \cdot r^{-1}) \cdot x \cdot r \cdot r^{-1}$$

$$= ((r \cdot x \cdot x \cdot 1) \cdot x \cdot 1) \cdot x \cdot 1 = ((r \cdot x \cdot x) \cdot x) \cdot x = r \cdot (((x \cdot x) \cdot x) \cdot x) = r \cdot x^4$$

Therefore, $f' \cdot r^{-1} = r \cdot x^4 \cdot r^{-1} = x^4 \cdot r \cdot r^{-1} = x^4 \cdot 1 = x^4 = f$. Since f' is in the Montgomery Canonical Form with respect to r, it is in general more efficient to compute $f' \cdot r^{-1}$, which itself is in the Montgomery Canonical Form, than it is to compute f directly.

Elliptic Curve Groups

An elliptic curve, G, is a mathematical group that is constructed over a specific field F, according to a specific set of rules that depend on the exact nature of F. In general, G is a subset of $F \times F$, and the $\cdot$ operation in G is defined in terms of the field operations $+$ and $\cdot$ on the

elements of F that constitute the ordered-pair elements of G. The two most commonly studied classes of elliptic curves are those constructed over fields belonging to $GF(p)$ or $GF(2^k)$.

Elliptic Curves over GF(p)

An elliptic curve over $GF(p)$ is defined by selected parameters a and b (both members of $GF(p)$) as the set of the all ordered pairs (x, y) that are solutions to the equation $y^2 = x^3 + ax + b$, where x, y are members of $GF(p)$ together with an extra point O, usually named as the point at infinity. It is assumed that p is a prime number greater than 3 and a, b in $GF(p)$ are selected such that $4a^3 - 27b^2 \neq 0$ in $GF(p)$. It has been well established that points on the elliptic curve and O form an Abelian group with respect to the following point addition rules:

Equations A

1. $O + O = O$

2. $(x, y) + O = (x, y)$

3. $(x, y) + (x, -y) = O$

4. Addition of two distinct points: $(x_1,y_1) + (x_2,y_2) = (x_3,y_3)$

$$L = (y_2 - y_1) \cdot (x_2 - x_1)^{-1}$$

$$x_3 = (L \cdot L) - x_1 - x_2$$

$$y_3 = L \cdot (x_1 - x_3) - y_1$$

5. Doubling of a point: $(x_1,y_1) + (x_1,y_1) = (x_3,y_3)$

$$L = (3x_1 \cdot x_1 + a) \cdot (2y_1)^{-1}$$

$$x_3 = (L \cdot L) - (2x_1)$$

$$y_3 = L \cdot (x_1 - x_3) - y_1$$

where the operations $\cdot$, $-$, $\cdot$, and inverse $(^{-1})$ are performed in the field $GF(p)$. The above rules define the method by which two points on the elliptic curve are "added" to get a third point.

These equations will be referred to in the future as Equation A.

Example

The elliptic curve equation $y^2 = x^3 + x + 1$ over the field $GF(23)$ will be illustrated. It turns out there are 28 points on the curve including the special point O. These points are

Point O

(0,1) and *(0,22)*

(1,7) and *(1,16)*

(3,10) and *(3,13)*

(4,0)

(5,4) and *(5,19)*

(6,4) and *(6,19)*

(7,11) and *(7,12)*

(9,7) and *(9,16)*

(11,3) and *(11,20)*

(12,4) and *(12,19)*

(13,7) and *(13,16)*

(17,3) and *(17,20)*

(18,3) and *(18,20)*

(19,5) and *(19,18)*

For example, *(1,7)* is on the elliptic curve because it satisfies the equation $y^2 = x^3 + x + 1$ in the field *GF(23)* (that is, modulo 23) because $7^2 = 1^3 + 1 + 1 \bmod 23$; $49 = 3 \bmod 23$; and $3 = 3 \bmod 23$.

The point addition of *(3,10)* and *(9,7)* is computed using arithmetic modulo 23, or the field arithmetic of *GF(23)*:

$L = (7-10) \cdot (9-3)^{-1} = (-3) \cdot 6^{-1} = (-3) \cdot 4 = -12 = 11$

$x_3 = (11 \cdot 11) - 3 - 9 = 121 - 12 = 109 = 17$

$y_3 = (11 \cdot (3-17)) - 10 = -164 = 20$

Therefore, the addition of $(3,10)$ and $(9,7)$ equals $(17,20)$. This example illustrates that the addition of two points on the curve using the above rules gives a third point on the curve.

Elliptic Curves over $GF(2^k)$

A non-supersingular elliptic curve over the field $GF(2^k)$ is defined by the parameters a and b in $GF(2^k)$, with $b \neq 0$, as the set of solutions (x, y) to the equation $y^2 + xy = x^3 + ax^2 + b$ together with the extra point O. This set of points form a group with respect to the addition rules:

Equations B

1. $O + O = O$
2. $(x, y) + O = (x, y)$
3. $(x, y) + (x, x+y) = O$
4. Addition of two distinct points: $(x_1,y_1) + (x_2,y_2) = (x_3,y_3)$

   $L = (y_1 + y_2) \cdot (x_1 + x_2)^{-1}$

   $x_3 = (L \cdot L) + L + x_1 + x_2 + a$

   $y_3 = (L \cdot (x_1 + x_3)) + x_3 + y_1$
5. Doubling of a point: $(x_1,y_1) + (x_1,y_1) = (x_3,y_3)$

   $x_3 = x_1 \cdot x_1 + b \cdot (x_1^{-1}) \cdot (x_1^{-1})$

   $y_3 = x_1 \cdot x_1 + (x_1 + y_1 \cdot x_1^{-1}) \cdot x_3 + x_3$

Point Multiplication

An elliptic curve cryptographic operation, whether it is an encryption, a decryption, a

signature, or a key-pass operation, always involves the computation of eP given e and P, where P is a point on the curve and e is a positive integer. The reverse of this operation, i.e., the computation of e given P and eP is known to be very difficult. This is called the elliptic curve discrete logarithm problem, for which no efficient algorithm is currently known.

Since the addition operation in the elliptic curve group, G, is defined using a series of field operations from the underlying field, F, given two points P and Q in G, computation of $P + Q$ or $P + P$ requires computation of a series of operations in the field. In particular, if $F \in GF(2^k)$, the equations 4 above show that computation of $P + Q$ requires one field inversion, three field multiplications, and nine field additions. On the other hand, the computation of $P + P$ requires one field inversion, three field multiplications and five field additions, as demonstrated by equations 5.

If e is about 500 bits in length, the number of elliptic curve operations (additions and doublings) necessary to calculate eP can be shown to be about 750. Each elliptic curve operation involves several (about 15–20) finite field operations. If the value of k (from $F \in GF(2^k)$) is also high (>100), these computations consume a significant amount of time, particularly in software. Therefore, fast hardware and software implementations of elliptic curve point multiplications are highly desirable in cryptography.

The following example in $GF(23)$ illustrates various approaches that can be taken towards optimizing the calculation of eP. Let $e = 18$ and $P = (3,10)$. Then $eP = 18(3,10)$ can be calculated by successively adding $(3,10)$ to itself 18 times using group addition as defined in equation A: $P + P + \ldots - P$ (18 copies of P), which requires 17 elliptic curve point addition operations.

However, there are faster algorithms known as "exponentiation methods," one example of which is a "binary method," shown below, which allows $18P$ to be computed as

Step 1: $(P) \cdot (P) = 2P$

Step 2: $(2P) + (2P) = 4P$

Step 3: $(4P) + (4P) = 8P$

Step 4: $(8P) + (8P) = 16P$

Step 5: $(16P) + (2P) = 18P$

Thus, only 5 point additions, or group operations, are utilized. The partial results as well as the final results are points on the curve as illustrated below:

Step 1: $P + P = 2P$: $(3,10) + (3,10) = (7,12)$

Step 2: $2P + 2P = 4P$: $(7,12) + (7,12) = (17,3)$

Step 3: $4P + 4P = 8P$: $(17,3) + (17,3) = (13,16)$

Step 4: $8P + 8P = 16P$: $(13,16) + (13,16) = (5,19)$

Step 5: $16P + 2P = 18P$: $(5,19) + (7,12) = (6,19)$

Thus, $18(3,10)$ is $(6,19)$. The elliptic curve discrete logarithm problem then becomes: knowing $(3,10)$ and $(6,19)$ and that $(6,19)$ is an integer multiple of $(3,10)$, what is this integer? The integer used for this example is equal to 18.

Given the binary representation of e as $e_{k-1}e_{k-2}...e_2e_1e_0$, the computation of eP can be accomplished using the binary method or any other M-ary method. For example, in order to compute $Q = eP$, the binary method proceeds as follows:

$Q := O$

for $i = k - 1$ downto 0

$Q := Q + Q$

if $e_i = 1$ then $Q := Q + P$

return Q.

Therefore, the computation of Q is performed by a series of elliptic curve point doublings $(Q := Q \cdot Q)$ and point additions $(Q := Q \cdot P)$.

SUMMARY OF THE INVENTION

The present invention optimizes the calculation of Elliptic Curve Cryptography computations through a transformation method that permits the use of any elliptic curve defined over any field F in a secure and efficient manner. The invention includes a method and apparatus for producing an elliptic curve point multiplication product, $Q = eP$. The invention utilizes an arbitrary integer e, and a point P on an elliptic curve group G defined over a field F, where the group G is a subset of the field F crossed with the field F. The present invention constructs a set G', a mapping T from G into the set G', a mapping T^{-1} from G' onto G, and an operation $\oplus$ defined on G', such that (a) given the point P, $T^{-1}(T(P)) = P$, and (b) $P+P = T^{-1}(P' \oplus P')$, where $P' = T(P)$. An elliptic curve point multiplication product Q is produced by transforming the point P to the point P' using the mapping T, performing the operation $\oplus$ on the point P' to determine the point $Q' = e\,P'$, and transforming the point Q' to the product Q using the mapping T^{-1}. The product Q is used in an elliptic curve cryptographic operation.

The present invention also includes a method for optimizing the calculation of cryptographic operations involving arbitrary expressions in finite field arithmetic through a transformation method that permits the use of any field F in an efficient manner. The invention includes a method for transforming any arbitrary finite calculation in any finite field into a canonical form in which other previously known algorithms can be applied, thereby achieving increased calculation speed and efficiency. The present invention teaches a set of transformations of the cryptographic calculations that allows the use of other known techniques that have only been applicable to certain limited special cases prior to this invention.

DETAILED DESCRIPTION OF THE INVENTION

The present invention provides a method for optimizing ECC computations for any curve in any field through focusing on one of the most computationally intense operations used in all ECC implementations, known as "elliptic curve point multiplication." Point multiplication requires the computation of eP, where P is a point in the elliptic curve and e is a positive integer. This operation is central to many elliptic curve cryptography functions, including encryption, decryption, random number generation, key-exchange, digital signing, and signature verification. The present invention achieves efficient ECC by providing a methodology for optimizing the implementation of the elliptic curve point multiplication operation. The present invention can be utilized to implement ECC over any curve in any field, including all individual member fields in $GF(p)$ and $GF(2^k)$.

The present invention further provides a methodology for optimizing computation of calculations involving a finite number of arbitrary field operations within any finite field. These calculations play a key role in computer implementations of numerous systems, including elliptic curve cryptosystems.

The present invention provides a "transformation method" which can be used to enable optimized implementations of elliptic curve cryptographic systems in hardware and software.

The present invention, because it employs a reversible transformation applied to the elements of the elliptic group, does not in any way alter the fundamental security properties of the mathematical algorithm used to perform the elliptic curve cryptography. The security of the overall ECC algorithm is determined by the choice of elliptic curve equations, number representation, arithmetic algorithms and other implementation aspects. As long as these choices are made according to reliable standards, the security of the implementation is not affected by use of the present invention.

The present invention can be used in any and all potential ECC applications, ranging from software for secure distribution of digital products such as movies and songs to hardware chips embedded in consumer electronic products such as cellular phones and smart cards. The cost-saving potential of the present invention can significantly enhance existing commercial applications and make previously infeasible business opportunities economically viable.

Section A

Given $G \subset F \times F$, an elliptic curve defined over the field F, the present invention provides an improved method to optimize the computation of eP, where e is an integer and P is an element of G.

The present invention includes:

(1) construction of a set G' and a method for representation of the members of G' in software and/or hardware;

(2) construction of and implementing an algorithm for a first mapping, T, from G into the set G' in software and/or hardware;

(3) construction of and implementing an algorithm for a second mapping, T^{-1}, which acts as the inverse of T, from G' onto G, in software and/or hardware; and

(4) construction of and implementing an algorithm, in software and/or hardware, for a set operation $\oplus$, defined in G'. For each invention, the following three conditions are satisfied:

(i) given any $P \in G$, then $T^{-1}(T(P)) = P$;

(ii) given any two points P and S in G, then $P + S = T^{-1}(P' \oplus S')$, where $P' = T(P)$ and $S' = T(S)$; and

(iii) G', T, $\oplus$, and T^{-1} and the corresponding algorithms are chosen such that given $P_1, P_2, \ldots, P_N \in G$, where N is an integer, computation of $T^{-1}(T(P_1) \oplus T(P_2) \oplus \ldots \oplus T(P_N))$ is in

general more optimized than computation of $P_1 + P_2 + \ldots + P_N$.

In other words, the present invention computes $Q = eP$ by first transforming the given point P to a transformed point P' using the algorithm for the first mapping T, then calculating the multiple sum $Q' = eP'$ using a "transformed," more computationally optimized version of the elliptic curve addition operation ($\oplus$) in the transformed domain, and finally transforming Q' back to Q using the algorithm for the second mapping, T^{-1}. Note that satisfaction of the conditions (i) and (ii) above ensures that this method can be applied to any point P belonging to G.

Under certain circumstances, when G', T, T^{-1}, and $\oplus$ are chosen carefully, it is possible to optimize computation of the point multiplication operation. Depending on the number of point additions to be performed, the additional cost of transforming the elements of G may or may not outweigh the improvements due to more optimized calculations in the transformed domain G'.

Section B

The present invention further provides a particular method for construction of G', T, and T^{-1} that can be applied to any elliptic curve group G.

Since G is a subset of $F \times F$, points in G can be written as ordered pairs (x, y), where x and y are elements of the field F. The present invention provides that a particular member r of F is first selected. The element r may be selected to be any member of the field F. Let t be the mapping from G into $F \times F$ that maps any point $P = (x, y)$ in G to some point $P' = (x', y')$ in G'. The present invention provides that $t(P) = t((x, y)) = (x \cdot r, y \cdot r) = P'$. Since x, y, and r are all members of F, so are $x \cdot r$ and $y \cdot r$. The present invention provides that G' is the image of G under t. In other words, G' is the set of all elements of $F \times F$ that have a point in G mapped to them by t. The present invention further provides that T is the transformation from G onto G' such that given any point P in G, then $T(P) = t(P) = P'$. While $P' = (x', y')$ is necessarily a member $F \times$

F, it is not necessarily a point in the elliptic curve group, G. P' can be obtained by computing $x' = x \cdot r$ and $y' = y \cdot r$.

Let Q' be any element of G'. Since $G' \subset F \times F$, we can write $Q' = (u', v')$, where u' and v' are members of F. Since G' is the image of G under t, then there must exist u and v, two elements of F, such that $(u, v) \in G$ and $u' = u \cdot r$ and $v' = v \cdot r$. Since u, v, and r are all members of the field F, then $u = u' \cdot r^{-1}$ and $v = v' \cdot r^{-1}$, where r^{-1} is the inverse of r under the multiplicative operation of F. Therefore, one can construct an inverse transformation $T^{-1}: G' \rightarrow G$ by letting T^{-1} map $Q' = (u', v')$, any element of G', to $(u' \cdot r^{-1}, v' \cdot r^{-1})$.

In formal terms, this more detailed embodiment of the present invention includes the steps of:

(1) constructing G' as the subset of $F \times F$ which is the image of G under the mapping $t: G \rightarrow F \times F$, where t is constructed by first selecting any element r of F, and then letting $t((x, y)) = (x \cdot r, y \cdot r)$, where $\cdot$ is the multiplicative operation in F;

(2) constructing the first mapping $T: G \rightarrow G'$ by letting $T(P) = t(P)$, where P is any point in G; and

(3) constructing the second mapping $T^{-1}: G' \rightarrow G$ by letting $T^{-1}((u', v')) = (u' \cdot r^{-1}, v' \cdot r^{-1})$, where (u', v') is any element of G'.

Given the above choices for G', T, and T^{-1}, it may be possible to optimize calculation of eP may through careful definition of a $\oplus$ operation in G' and careful selection of r. Certain values of r, for instance, may provide faster software implementations, while others may enable more algorithmic parallelism.

Section C

Another detailed embodiment of the present invention applies the methods of Sections A

and B to the elliptic curves defined over the specific fields belonging to $GF(p)$. In this embodiment, a new transformed operation $\oplus$ is constructed such that conditions (i), (ii), and (iii) in Section A are satisfied.

The present invention includes a method for optimizing calculations of eP when F is an individual member field of $GF(p)$. In this embodiment, G is an elliptic curve group over $F \in GF(p)$, and G', T, T^{-1} are constructed in accordance with the method of the invention described in Section B, above, through choosing an arbitrary element r of F. The present invention constructs a "transformed" operation $\oplus$ in G' as follows. Given any two elements of G', (x_1', y_1') and (x_2', y_2'), then the present invention defines $(x_1', y_1') \oplus (x_2', y_2')$ to be given by (x_3', y_3'), where

Equations A'

$$z' = (x_2' - x_1')^{-1} \cdot r^2$$

$$L' = (y_2' - y_1') \cdot z' \cdot r^{-1}$$

$$x_3' = L' \cdot L' \cdot r^{-1} - x_1' - x_2'$$

$$y_3' = L' \cdot (x_1' - x_3') \cdot r^{-1} - y_1'$$

Using the above definition of $\oplus$ for the operation of the "addition" of elements of G', the present invention derives the following set of field equations for the operation of adding any "point" (x_1', y_1') in G' to itself:

$(x_1', y_1') \oplus (x_1', y_1') = (x_3', y_3')$, where

$$z' = (y_1' + y_1')^{-1} \cdot r^2$$

$$L' = ((x_1' + x_1' + x_1') \cdot x_1' \cdot r^{-1} + a') \cdot z' \cdot r^{-1}$$

$$x_3' = L' \cdot L' \cdot r^{-1} - x_1' \cdot x_1'$$

$$y_3' = L' \cdot (x_1' - x_3') \cdot r^{-1} - y_1'$$

It is now shown how the present invention ensures that G', T, T^{-1}, and $\oplus$ together satisfy

conditions (i), (ii), and (iii) set forth in Section A.

(i) Let P be any point in the elliptic curve group G. Then there exist some elements x and y of F such that $P = (x, y)$. Then $T^{-1}(T(P)) = T^{-1}(T((x, y))) = T^{-1}((x \cdot r, y \cdot r)) = (x \cdot r \cdot r^{-1}, y \cdot r \cdot r^{-1}) = (x, y) = P$. Therefore, Condition (i) in Section A is satisfied.

(ii) Given any two points P and S in G, then we need to show that $P + S = T^{-1}(P' \oplus S')$, where $P' = T(P)$ and $S' = T(S)$. Let $P = (x_1, y_1)$, $P' = (x_1', y_1')$, $S = (x_2, y_2)$, $S' = (x_2', y_2')$, $Q = P - S = (x_3, y_3)$ and $Q' = P' + S' = (x_3', y_3')$. Then, applying the rules for point addition in the elliptic curve group given by Equations A, the coordinates of Q are given by $x_3 = L \cdot L - x_1 - x_2$ and $y_3 = L \cdot (x_1 - x_3) - y_1$, where $L = (y_2 - y_1) \cdot z$, and $z = (x_2 - x_1)^{-1}$. Equations A' above, on the other hand, give the coordinates for Q'. As such, it can be shown that,

$$\begin{aligned} z' &= (x_2' - x_1')^{-1} \cdot r^2 \\ &= (x_2 \cdot r - x_1 \cdot r)^{-1} \cdot r^2 \\ &= (x_2 - x_1)^{-1} \cdot r \\ &= z \cdot r \end{aligned}$$

$$\begin{aligned} L' &= (y_2' - y_1') \cdot z' \cdot r^{-1} \\ &= (y_2 \cdot r - y_1 \cdot r) \cdot (z \cdot r) \cdot r^{-1} \\ &= (y_2 - y_1) \cdot z \cdot r \\ &= L \cdot r \end{aligned}$$

$$\begin{aligned} x_3' &= L' \cdot L' \cdot r^{-1} - x_1' - x_2' \\ &= (L \cdot r) \cdot (L \cdot r) \cdot r^{-1} - x_1 \cdot r - x_2 \cdot r \\ &= (L \cdot L - x_1 - x_2) \cdot r \\ &= x_3 \cdot r \end{aligned}$$

$$\begin{aligned} y_3' &= L' \cdot (x_1' - x_3') \cdot r^{-1} - y_1' \\ &= (L \cdot r) \cdot (x_1 \cdot r - x_3 \cdot r) \cdot r^{-1} - y_1 \cdot r \\ &= (L \cdot (x_1 - x_3) - y_1) \cdot r \\ &= y_3 \cdot r \end{aligned}$$

Therefore, $(x_3', y_3') = (x_3 \cdot r, y_3 \cdot r)$, which implies that $P' \oplus S' = T(P + S)$, as required. Hence, Condition (ii) in Section A is satisfied.

(iii) The present invention has provided a method for the selection of G', T, $\oplus$, and T^{-1} and

their corresponding algorithms in a manner such that given $P_1, P_2, \ldots, P_N \in G$, where N is an integer, computation of $T^{-1}(T(P_1) \oplus T(P_2) \oplus \ldots \oplus T(P_N))$ is in general more optimized than computation of $P_1 + P_2 + \ldots + P_N$. To verify this, note that calculation of $T(P_1) \oplus T(P_2) \oplus \ldots \oplus T(P_N)$ involves repeated application of the expressions in Equations A'. Note, however, that these expressions are in the Montgomery Canonical Form with respect to r. As such, the Montgomery Algorithm in GF(p) can be readily applied to the calculation of $T(P_1) \oplus T(P_2) \oplus \ldots \oplus T(P_N)$ to create an optimized hardware and/or software implementation. Therefore, Condition (ii) in Section A is satisfied.

Section D

Another detailed embodiment of the present invention applies the methods of Sections A and B to the elliptic curves defined over the specific fields belonging to $GF(2^k)$. In this embodiment, a new transformed operation $\oplus$ is constructed such that conditions (i), (ii), and (iii) in Section A are satisfied.

The present invention further includes a method for optimizing calculations of eP when F is an individual member field of $GF(2^k)$. In this embodiment, G is an elliptic curve group over $F \in GF(2^k)$, and G', T, T^{-1} are constructed in accordance with the method of the invention described in Section B, above, through choosing an arbitrary element r of F. The present invention constructs a "transformed" operation $\oplus$ in G' as follows. Given any two elements of G', say (x_1', y_1') and (x_2', y_2'), then the present invention defines $(x_1', y_1') \oplus (x_2', y_2')$ to be given by (x_3', y_3'), where

Equations B'

$z' = (x_1' \cdot x_2')^{-1} \cdot r^2$

$$L' = (y_1' - y_2') \cdot z' \cdot r^{-1}$$

$$x_3' = (L' \cdot L' \cdot r^{-1}) + L' + x_1' + x_2' + a'$$

$$y_3' = (L' \cdot (x_1' + x_3') \cdot r^{-1}) + x_3' + y_1'$$

Using the above definition of $\oplus$ for the operation of the addition of points in G', the present invention derives the following set of field equations for the operation of "doubling a point", *i.e.* adding a point (x_1', y_1') in G' to itself:

$(x_1', y_1') \oplus (x_1', y_1') = (x_3', y_3')$, where

$$z' = (x_1')^{-1} \cdot r^2$$

$$x_3' = x_1' \cdot x_1' \cdot r^{-1} + (z' \cdot z' \cdot r^{-1}) \cdot b' \cdot r^{-1}$$

$$y_3' = x_1' \cdot x_1' \cdot r^{-1} + (x_1' + y_1' \cdot z' \cdot r^{-1}) \cdot x_3' \cdot r^{-1} - x_3'$$

It is now shown how the present invention ensures that G', T, T^{-1}, and $\oplus$ together satisfy conditions (i), (ii), and (iii) set forth in Section A.

(i) Let P be any point in the elliptic curve group G. Then there exist some elements x and y of F such that $P = (x, y)$. Then $T^{-1}(T(P)) = T^{-1}(T((x, y))) = T^{-1}((x \cdot r, y \cdot r)) = (x \cdot r \cdot r^{-1}, y \cdot r \cdot r^{-1}) = (x, y) = P$. Therefore, Condition (i) in Section A is satisfied.

(ii) Given any two points P and S in G, then we need to show that $P + S = T^{-1}(P' \oplus S')$, where $P' = T(P)$ and $S' = T(S)$. Let $P = (x_1, y_1)$, $P' = (x_1', y_1')$, $S = (x_2, y_2)$, $S' = (x_2', y_2')$, $Q = P + S = (x_3, y_3)$ and $Q' = P' + S' = (x_3', y_3')$. Then, applying the rules for point addition in the elliptic curve group given by Equations A, the coordinates of Q are given by $x_3 = L \cdot L + L + x_1 + x_2 + a$ and $y_3 = L \cdot (x_1 + x_3) + x_3 + y_1$, where $L = (y_1 + y_2) \cdot z$, and $z = (x_1 + x_2)^{-1}$. Equations A' above, on the other hand, give the coordinates for Q'. As such, it can be shown that,

$$z' = (x_1' \cdot x_2')^{-1} \cdot r^2$$
$$= (x_1 \cdot r \cdot x_2 \cdot r)^{-1} \cdot r^2$$

$$= (x_1 + x_2)^{-1} \cdot r$$
$$= z \cdot r$$

$$L' = (y_1' + y_2') \cdot z' \cdot r^{-1}$$
$$= (y_1 \cdot r + y_2 \cdot r) \cdot (z \cdot r) \cdot r^{-1}$$
$$= (y_1 + y_2) \cdot z \cdot r$$
$$= L \cdot r$$

$$x_3' = L' \cdot L' \cdot r^{-1} + L' + x_1' + x_2' + a'$$
$$= (L \cdot r) \cdot (L \cdot r) \cdot r^{-1} + L \cdot r + x_1 \cdot r + x_2 \cdot r + a' \cdot r$$
$$= (L \cdot L + L + x_1 + x_2 + a') \cdot r$$
$$= x_3 \cdot r$$

$$y_3' = L' \cdot (x_1' + x_3') \cdot r^{-1} + x_3' + y_1'$$
$$= (L \cdot r) \cdot (x_1 r + x_3 r) \cdot r^{-1} + x_3 \cdot r + y_1 \cdot r$$
$$= (L \cdot (x_1 + x_3) + x_3 + y_1) \cdot r$$
$$= y_3 \cdot r$$

Therefore, we have $(x_3', y_3') = (x_3 \cdot r, y_3 \cdot r)$ which implies that $P' \oplus S' = T(P + S)$, as required. Hence, Condition (ii) in Section A is satisfied.

(iii) The present invention has provided a method for the selection of G', T, $\oplus$, and T^{-1} and their corresponding algorithms in a manner such that given $P_1, P_2, \ldots, P_N \in G$, where N is an integer, computation of $T^{-1}(T(P_1) \oplus T(P_2) \oplus \ldots \oplus T(P_N))$ is in general more optimized than computation of $P_1 + P_2 + \ldots + P_N$. To verify this, note that calculation of $T(P_1) \oplus T(P_2) \oplus \ldots \oplus T(P_N)$ involves repeated application of the expressions in Equations B'. Note, however, that these expressions are in the Montgomery Canonical Form with respect to r. As such, the Montgomery Algorithm in $GF(2^k)$ can be readily applied to the calculation of $T(P_1) \oplus T(P_2) \oplus \ldots \oplus T(P_N)$ to create an optimized hardware and/or software implementation. Therefore, Condition (ii) in Section A is satisfied.

Section E

The present invention further provides a method for achieving higher efficiencies when

utilizing the methods of Sections C and D above by providing specific choices of r.

The present invention works with any element r in the field F over which the elliptic curve group G is defined. The exact choice of the element r, however, affects the computational characteristics of the resulting calculations. The present invention teaches that the selection of r can optimize specific aspects of a software and/or hardware implementation within specific computer environments. For instance, choosing r to be a multiple of 32 can have beneficial effects on 32-bit computers. Given a particular selection of r, the calculation of $a \cdot b \cdot r^{-1}$ may be done in more than one way, some of which may be more computationally efficient. The following selections of r are preferred:

1. Field $GF(p)$: r is selected as the smallest power of 2 that is larger than p.

2. Field $GF(p)$: r can be selected as the product of k prime numbers, which gives the resulting algorithm a high degree of parallelism.

3. Field $GF(2^k)$: r is selected as $x^k \bmod n(x)$, where $n(x)$ is the irreducible polynomial generating the field $GF(2^k)$.

Other selections of r for different fields are also possible. The transformation algorithms work independently of this selection.

Section F

The present invention further provides a method for optimizing calculation of a finite number of arbitrary field operations over any finite field. Let f be a valid expression defined within F involving a finite number of variables, and a finite number of the field operations $+$, $\cdot$, $-$, and $^{-1}$. The present invention provides a method for optimizing computation of f, which includes carrying out the following steps in sequence:

(1) Select r to be any single element of the field F. The element r, a constant, will be used to

transform the expression f into a new expression f' through applying a series of substitutions in accordance with the substitution technique described earlier in this document. If the expression f already contains a constant or variable denoted by the symbol r, then rename the symbol r in this step-by-step procedure to some unique value, and interpret the subsequent steps of this procedure as if r were renamed appropriately in them. Note that the expression f may coincidentally contain constants or variables that may have the same field value as the selected element r, without affecting this procedure. Subsequent steps of this procedure will rely on the expression f being initially free of "primed" symbols such as d' or j''. If the expression f initially contains any variables or constants which are denoted by "primed" symbols, then replace each primed variable or constant symbol with a unique unprimed name. Subsequent substitution steps of this procedure will employ source expressions containing primed symbols, which by convention in this patent are not allowed to match symbols that are not primed. Note that source expressions containing unprimed symbols, such as x, are allowed by convention in this patent to match variable symbols or constant symbols which may be either primed or unprimed.

(2) Transform the expression f into the expression f_1 by replacing all occurrences of the source expression x with the target expression x'. In this substitution, x denotes a variable or constant occurring in the expression f. This replaces all variables and constants with primed symbols. Note that this occurs without affecting any coefficients that may exist in the expression f.

(3) Transform the expression f_1 into the expression f_2 by replacing all occurrences of the source expression $x \cdot y$ with the target expression $x \otimes y$. In this substitution, x and y denote subexpressions of f_1, which should contain only primed symbols, and $\otimes$ is used as

an alternate symbol to represent the multiplication operation in the field F. The purpose of this step is to label as $\otimes$ all of the original $\cdot$ operators occurring in the expression f, to distinguish them from the $\cdot$ operators that will be introduced into the transformed expression during the following steps of this method.

(4) Transform the expression f_2 into the expression f_3 by replacing all occurrences of the source expression x^{-1} with the target expression $x^{-1} \cdot r^2$. In this substitution, x denotes a subexpression of f_2.

(5) Transform the expression f_3 into the expression f_4 by replacing all occurrences of the source expression $x \otimes y$ with the target expression $x \cdot y \cdot r^{-1}$. In this substitution, x and y denote subexpressions of f_3.

(6) Transform the expression f_4 into the expression f' by replacing all occurrences of the source expression x' with the target expression $x \cdot r$. In this substitution, x' denotes a primed variable or primed constant occurring in the expression f_3, specifically excluding all instances of the unprimed constant r. The effect of this step is to replace every primed symbol with its unprimed form multiplied by r.

Upon completion of the above steps, the expression f is transformed into a new expression f'. The present invention has carefully specified the preceding steps in such a way as to ensure that $f = f' \cdot r^{-1}$. To prove this, the result is demonstrated for the four special cases when $f = x + y$, $f = x - y$, $f = x \cdot y$, and $f = x^{-1}$, where x and y are elements in the field F. The general result follows from the commutative, associative and distributive properties of the field.

Case 1. Transformed Addition

Let $f = x + y$. Applying the substitution method of the present invention to the expression f results in the transformed expression $f' = x \cdot r + y \cdot r$. To see that $f = f' \cdot r^{-1}$, note that $f = x + y$

$= (x - y) \cdot r \cdot r^{-1} = (x \cdot r - y \cdot r) \cdot r^{-1} = f' \cdot r^{-1}$.

Case 2. Transformed Subtraction

Let $f = x - y$. Applying the substitution method of the present invention to the expression f results in the transformed expression $f' = x \cdot r - y \cdot r$. To see that $f = f' \cdot r^{-1}$, note that $f = x - y = (x - y) \cdot r \cdot r^{-1} = (x \cdot r - y \cdot r) \cdot r^{-1} = f' \cdot r^{-1}$.

Case 3. Transformed Multiplication

Let $f = x \cdot y$. Applying the substitution method of the present invention to the expression f results in the transformed expression $f' = (x \cdot r) \cdot (y \cdot r) \cdot r^{-1}$. To see that $f = f' \cdot r^{-1}$, note that $f = x \cdot y = x \cdot y \cdot r^2 \cdot r^{-2} = x \cdot y \cdot r \cdot r \cdot r^{-1} \cdot r^{-1} = x \cdot r \cdot y \cdot r \cdot r^{-1} \cdot r^{-1} = (x \cdot r) \cdot (y \cdot r) \cdot r^{-1} \cdot r^{-1} = f' \cdot r^{-1}$.

Case 4. Transformed Inversion

Let $f = x^{-1}$. Applying the substitution method of the present invention to the expression f results in the transformed expression $f' = (x \cdot r)^{-1} \cdot r^2$. To see that $f = f' \cdot r^{-1}$, note that $f = x^{-1} \cdot r^2 \cdot r^{-2} = x^{-1} \cdot r^{-1} \cdot r^2 \cdot r^{-1} = (x \cdot r)^{-1} \cdot r^2 \cdot r^{-1} = f' \cdot r^{-1}$.

Thus, the present invention provides a method to transform any expression f involving a finite number of field operations within a finite field F into the form $f' \cdot r^{-1}$. Furthermore, the expression $f' \cdot r^{-1}$ constructed by the present invention is guaranteed to be in the Montgomery Canonical Form. To verify this, note that (i) the substitution steps of the method of the present invention ensure that if the original expression f includes any subexpressions that are of the form $x \cdot y$, such subexpressions are transformed into the form $(x \cdot r) \cdot (y \cdot r) \cdot r^{-1}$, which is in the

Montgomery Canonical Form; and (ii) whenever the substitution steps of the method of the present invention introduce a new multiplication operation into the transformed expression, such operation brings with it a single additional operand which is always a power of r, thus preserving the Montgomery Canonical Form of the subexpression it is introduced into.

Depending on the exact nature of F, and the number of the multiplication operations in the expression f and the exact number and nature of the operations involved in the calculation of f', computation of the expression $f' \cdot r^{-1}$ may be more efficient than direct computation of the expression f. This is particularly likely, when the field F is a member of either $GF(p)$ or $GF(2^k)$. For, in such instances, the Montgomery Algorithm can be applied to the expression $f' \cdot r^{-1}$ to ensure optimized computation of the value that the expression f evaluates to.

Section G

The present invention may also be used with "projective coordinates," which are used to eliminate the need for performing inversion.

For example, in projective coordinates, a point on the elliptic curve group G has 3 coordinate values: (x_1, y_1, z_1) while the affine coordinates requires only two values: (x_1, y_1).

For example, for elliptic curves defined over $GF(2^k)$, given the distinct points P and Q expressed in projective coordinates:

$$P := (x_1, y_1, z_1)$$

$$Q := (x_2, y_2, z_2)$$

the projective coordinates of the sum of 2 points on the elliptic curve are:

$$P + Q := (x_3, y_3, z_3)$$

using the following addition rules:

$$A = x_2 \cdot z_1 - x_1$$

$$B = y_2 \cdot z_1 + y_1$$

$$C = A + B$$

$$D = A^2 \cdot (A + a \cdot z_1) + z_1 \cdot B \cdot C$$

$$x_3 = A \cdot D$$

$$y_3 = C \cdot D + A^2 \cdot (B \cdot x_1 + A \cdot y_1)$$

$$z_3 = A^3 \cdot z_1$$

This computation requires 13 field multiplications, and no inversions.

Similarly, the addition formulae for computing $2P$ is given as:

$$A = x_1 \cdot z_1$$

$$B = b \cdot z_1^4 + x_1^4$$

$$x_3 = A \cdot B$$

$$y_3 = x_1^4 \cdot A + B \cdot (x_1^2 + y_1 \cdot z_1 + A)$$

$$z_3 = A^3$$

This computation requires 7 field multiplications, and no inversions.

Thus, the use of projective coordinates eliminate the inversions at the expense of storing 3 $GF(2^k)$ values to represent P and performing a few more multiplications.

The present invention can also be used in conjunction with projective coordinates. The addition rules would then be modified as follows:

$$A' = x_2' \cdot z_1' \cdot r^{-1} + x_1'$$

$$B' = y_2' \cdot z_1' \cdot r^{-1} + y_1'$$

$$C' = A' + B'$$

$$D' = (A' \cdot A' \cdot r^{-1}) \cdot (A' + a' \cdot z_1' \cdot r^{-1}) \cdot r^{-1} + (z_1' \cdot B' \cdot r^{-1}) \cdot C' \cdot r^{-1}$$

$$x_3' = A' \cdot D' \cdot r^{-1}$$

$$y_3' = C' \cdot D' \cdot r^{-1} + (A' \cdot A' \cdot r^{-1}) \cdot (B' \cdot x_1' \cdot r^{-1} + A' \cdot y_1' \cdot r^{-1}) \cdot r^{-1}$$

$$z_3' = ((A' \cdot A' \cdot r^{-1}) \cdot A' \cdot r^{-1}) \cdot z_1' \cdot r^{-1}$$

Similarly the rules for computing 2P are modified as

$$A' = x_1' \cdot z_1' \cdot r^{-1}$$

$$B' = (((b' \cdot z_1' \cdot r^{-1}) \cdot z_1' \cdot r^{-1}) \cdot z_1' \cdot r^{-1}) \cdot z_1' \cdot r^{-1} + ((x_1' \cdot x_1' \cdot r^{-1}) \cdot x_1' \cdot r^{-1}) \cdot x_1' \cdot r^{-1}$$

$$x_3' = A' \cdot B' \cdot r^{-1}$$

$$y_3' = (((x_1' \cdot x_1' \cdot r^{-1}) \cdot x_1' \cdot r^{-1}) \cdot x_1' \cdot r^{-1}) \cdot A' \cdot r^{-1} +$$

$$B' \cdot (x_1' \cdot x_1' \cdot r^{-1} + y_1' \cdot z_1' \cdot r^{-1} + A') \cdot r^{-1}$$

IMPLEMENTATION

The present invention may be implemented on any conventional or general purpose PC computer system. It may also be used in conjunction with any network system, including the Internet. A preferred embodiment of a computer system for implementing this invention is an Intel Pentium II PC 233 MHz, running Windows NT 4.0.

The present invention can be implemented in any programming language including C and Java. The following are examples of pseudo code suitable for implementing the present invention.

```
-----------------------------------------------------------------
Setup:  a, b : Parameters of the elliptic curve (EC)
        F : The field upon which the EC is based
            Either GF(p) or GF(2^k)
        * : field multiplication
        + : field addition
        - : field subtraction
        -1 : field inversion
        P=(P(x), P(y)) : A point on the EC
        P(x) & P(y) are affine coordinates
-----------------------------------------------------------------
Algorithm Identifier:   ExpPoint
Input:  e : k-bit integer
        P : Point on the EC, P = (P(x), P(y))
Output: Q : Point on the EC, Q = (Q(x), Q(y))
        Q := eP = (P+P+...+P) (e times P)
function ExpPoint
begin
  /* Transform P to P' using r */
  P'(x) = P(x) * r
  P'(y) = P(y) * r
  /* Start with O'  point at infinity */
  Q' = O'
```

```
    /* Binary method loop */
    for i=k-1 downto 0 do
        Q' := DoublePoint(Q')
        if e_i=1 then Q' := AddPoint(Q', P')
    /* Transform Q' to Q using r */
    Q(x) = Q'(x) * r^-1
    Q(y) = Q'(y) * r^-1
    return Q
end
-----------------------------------------------------------------
Algorithm Identifier:  AddPoint
Input:  P' : Transformed Point on the EC
        Q' : Transformed Point on the EC
Output: T' : Transformed Point on the EC
        T' := P' + Q' using the EC point addition rules
function AddPoint
begin
/* If the underlying field is GF(p)      */
  lambda' = Multiply((Q'(y) - P'(y)), Inverse(Q'(x) - P'(x))
  T'(x) = Multiply(lambda', lambda') - P'(x) - Q'(x)
  T'(y) = Multiply(lambda', (P'(x) - T'(x))) - P'(y)
  return T
/* If the underlying field is GF(2^k)   */
  lambda' = Multiply((P'(y) + Q'(y)), Inverse(P'(x) + Q'(x)))
  T'(x) = Multiply(lambda', lambda') + lambda' + P'(x) + Q'(x) +
a'
  T'(y) = Multiply(lambda', (P'(x) + T'(x))) + T'(x) + P'(y)

  return T
end
-----------------------------------------------------------------
Algorithm Identifier:  DoublePoint
Input:  P' : Transformed Point on the EC
Output: T' : Transformed Point on the EC
        T' := P' + P  using the EC point doubling rules
function DoublePoint
begin
/* If the underlying field is GF(p)              */
  lambda' = Multiply(Multiply(3P'(x),P'(x)) + a'),
Inverse(2P'(y)))
  T'(x) = Multiply(lambda', lambda') - 2P'(x)
  T'(y) = Multiply(lambda', (P'(x) - T'(x))) - P'(y)
  return T
/* Else if the underlying field is GF(2^k)      */
  T'(x) = Multiply(P'(x), P'(x)) +
          Multiply(b', Multiply(Inverse(P'(x), P'(x))))
  T'(y) = Multiply(P'(x), P'(x)) +
          Multiply(P'(x) + Multiply(P'(y), Inverse(P'(x))),
T'(x)) + T'(x)
  return T
end
-----------------------------------------------------------------
Algorithm Identifier:  Inverse
Input :  u: Field element
Output:  t: Field element
function Inverse
begin
  t = u^-1  * r^2
  return t
```

```
end
-----------------------------------------------------------------
Algorithm Identifier:  Multiply
Input :  u, v: Field elements
Output:  t: Field element
function Multiply
begin
  t = u * v * r^-1
  return t
end
-----------------------------------------------------------------
```

A number of references describe the mathematical background for the present invention. Those references include: P. L. Montgomery, Modular multiplication without trial division, "Mathematics of Computation," 44(170):519–521, April 1985; D. E. Knuth, "The Art of Computer Programming: Seminumerical Algorithms," volume 2, Second edition, Reading, MA: Addison–Wesley, 1981; C. K. Koc and T. Acar, Montgomery multiplication in GF(2^k), "Proceedings of Third Annual Workshop on Selected Areas in Cryptography," pages 95–106, Queen's University, Kingston, Ontario, Canada, August 15–16, 1996; C. K. Koc and T. Acar., Fast software exponentiation in GF(2^k), "Proceedings, 13th Symposium on Computer Arithmetic," pages 225–231, Asilomar, California, July 6–9, 1997, Los Alamitos, CA: IEEE Computer Society Press; J. C. Bajard, L. S. Didier, and P. Kornerup, An RNS Montgomery multiplication algorithm, "Proceedings, 13th Symposium on Computer Arithmetic," pages 234–239, Asilomar, California, July 6–9, 1997, Los Alamitos, CA: IEEE Computer Society Press; D. R. Stinson. "Cryptography Theory and Practice," CRC Press, 1995; V. Miller, Uses of elliptic curves in cryptography, "Advances in Cryptology – CRYPTO 85, Proceedings," pages 417–426, New York, NY: Springer–Verlag, 1985; N. Koblitz, Elliptic curve cryptosystems, "Mathematics of Computation," 48:203–209, 1987; N. Koblitz, "A Course in Number Theory and Cryptography," New York, NY: Springer–Verlag, 1987; A. J. Menezes, "Elliptic Curve Public Key Cryptosystems," Boston, MA: Kluwer Academic Publishers, 1993; R.L. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public-key Cryptosystems,"

Communications of the ACM, 21(2):120–126, 1978; T. Beth, M. Frisch, and G.J. Simmons, Public-key Cryptography: State of the Art and Future Directions. Springer–Verlag, NY, 1991; IEEE Working Group P1363, Working Draft: IEEE 1363: Standard for RSA, Diffie–Hellman and Related Public-key Cryptography. In preparation, 1995; RSA Laboratories, Answers to Frequently Asked Questions about Today's Cryptography. Version 3.0, 1996; G.B. Agnew, R.C. Mullin, I.M. Onyszchuk, and S.A. Vanstone, An implementation of a fast public-key cryptosystem. Journal of Cryptology, 3(2):63–79, 1991. All of these publications are herein incorporated by reference as if each individual publication were specifically and individually set forth herein.

Having described and illustrated the principles of our invention with reference to a preferred embodiment, it will be apparent that the invention can be modified in arrangement and detail without departing from such principles. As such, it should be recognized that the detailed embodiment is illustrative only and should not be taken as limiting the scope of our invention. Rather, we claim as our invention all such embodiments as may fall within the scope and spirit of the following claims and equivalents thereto.